Improving Student's Learning and Cooperation Skills Using Coding Dojos (In the Wild!)

Caio Matheus Campos de Oliveira, Edna Dias Canedo, Henrique Faria, Luis Henrique Vieira Amaral, Rodrigo Bonifácio
Computer Science Department
University of Brasília - (UnB), P.O. Box 4466, 70910-900
Brasília-DF, Brazil
caiomcoliveira@gmail.com, ednacanedo@unb.br, hfaria020@gmail.com, luis.amaralh@gmail.com, rbonifacio@unb.br

***Abstract*—Collaborative development approaches (e.g., pair-programming, coding dojo, and hackathons) have gained increasing attention in recent years, mostly because they help to share knowledge during software development activities and might shorten development cycles and increase the quality of software products. Collaborative development approaches bring also the potential benefit to contribute to learning activities. For instance, novices might participate on collaborative development sessions in order to learn new development practices, tools, and techniques used in a software development project. Besides these potential benefits, little is known about the perception of students engaged in collaborative development efforts. Therefore, in this paper we investigate whether or not the engagement of students in collaborative development efforts contributes to the learning process of software development practices and techniques, as well as the perceived benefits and challenges related to collaborative software development activities. To this end, we first performed several of coding dojo sessions during a period of 18 months. These development sessions have been conducted within the context of a real software modernization effort, which aims to modernize two enterprise systems of the Brazilian Army. After that, we carried out a qualitative study where the participants (students, software developers, and software architects) answered a survey, in order to understand the learning benefits of using coding dojo in software development activities. The results so far are encouraging. Coding Dojos allowed professors and software architects to seamless share their experience in software development with the students. According to the answers, the methodology created a better environment for the team, allowing better discussions and ideas to be shared and implemented. This has helped the team members to solve problems easier than by themselves, bringing additional benefits, such as steep the learning curve in programming languages, usage of development tools, understanding the requirements, and code refactoring.**

***Index Terms*—Coding Dojo, collaborative learning, collaborative software development, empirical studies in software engineering.**

I. Introduction

Pair Programming and Coding Dojo are two practices that have being increasingly adopted for cooperative software development. While the former encourages the collaboration in pairs, the latter promotes collaboration in groups. The use of collaborative practices is particularly promising for acquiring programming skills, when part of the development team has little or no practical experience with software development [1]. The use of these collaborative practices bring several benefits, such as increasing product quality and shortening development cycles. Besides that, the use of collaborative practices might also help on the learning process of programming languages, architectural styles, and software development practices and techniques. This benefit arises because the team of participants often involve expert and novice developers during coding dojo sessions [2].

For this reason, the use of coding dojos seems to be particularly promising to teach undergraduate students in subjects related to software engineering, and existing research works report on these benefits using academic settings [1]. For instance, Estácio et al. conducted a controlled experiment to investigate how effective coding dojo is to help on the learning process of programming languages [1]. Similarly, Rodrigues et al. describe their experience using coding dojo sessions in a classroom, with the specific aim of teaching programming language courses [3]. That is, none of these studies report on the use of coding dojo as a teaching vehicle in "the wild", using real software development projects during a long-lasting time frame.

In this paper, we investigate the relevance of coding dojos to prepare students to start their careers as software engineers, by considering several of dojo sessions carried out during a real software modernization project of two non-trivial enterprise systems from the Brazilian Army. The first system (Material Endowment System – SISDOT) deals with the distribution of materials and equipments to all organizational units of the Brazilian Army (considering well-defined rules of distribution). This is a Java Enterprise Edition system with more than 40 thousand lines of code (KLOC).

The second system (Bulletin System – SISBOL) is responsible for managing the official communication of different types of events within the Brazilian Army. It actually involves the implementation of a configurable workflow for creating notes and composing official documents that must follow

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

organization-specific processes for approval and (internal) disclosure. SISBOL is also an enterprise system that has been implemented using a service-oriented architecture [4], where the front-end is based on AngularJS [5] and the back-end is based on Java Enterprise Edition. The current implementation comprises almost 20 KLOC.

Most of the development activities related to this modernization effort were conducted using a collaborative approach (either pair-programming or coding dojo), and the dojo sessions occurred at least once a week. In most cases, one or two expert developers lead the dojo sessions, which often involve at least three students with different degrees of knowledge as participants. Altogether, the main contributions of this paper are:

- An experience report on the use of collaborative practices (particularly coding dojo) during a real software modernization effort
- The results of an empirical study that aims to investigate whether or not the involvement with coding dojo sessions help students to increase their knowledge in subjects related to software engineering (including software design and implementation, software testing, and configuration management) and collaborative work.

II. Background and Related Work

In this section we first introduce the core ideas related to Coding Dojo (Section II-A) and the use of Coding Dojo to support the learning process of programming and software engineering (Section II-B). After that, we characterize the theoretical pedagogical framework of Coding Dojo (Section II-C).

A. Coding Dojo

The main goals of Coding Dojo is to promote a learning process through collaboration and to build an environment that leads to a non-competitive and inclusive experience [6]. Furthermore, this practice presents many aspects related to sociability, since certain behaviors are put in practice within an environment in which students and practitioners work and learn together [7].

To drive a Coding Dojo session, it is necessary to define a software development problem (or programming goal) and then put the necessary skills into practice to solve it. During the activities, it is possible to learn together and contribute to solve the problems even without having all the necessary skills—assuming that in each execution session one expert and one beginner work together [8]. Once the range of experienced situations increase, the student can be able to expand their technical knowledge and start to resolve some challenge and become an expert [6]. A single room with enough space to support all the participants is the ideal environment to achieve a productive Dojo in a meeting. That place usually requires only a projector, a computer or a laptop and a white-board space for sketching, explaining and designing their discussions [8].

Randori and *Prepared Kata* are two popular formats for conducting Coding Dojo. Considering the Randori approach, a Coding Dojo session is organized in turns that are led by a pilot and a co-pilot, which work together with the participants to solve the problems. At the end of each turn, the pilot joins the audience, the co-pilot becomes the pilot, and a new participant from the audience becomes the "new co-pilot" [6]. In contrast to the Randori format, in the Prepared Kata at least one of the participates must have solved the Kata prior to the meeting. This lead present all the steps (s)he follow to solve the challenge, while the other participants ask questions and make suggestions [6].

In this research, we follow a variant of the Randori style, where we do not consider the co-pilot role—each session is only led by a practitioner or a student. Figure 1 shows two recent sessions of Coding Dojo we conducted in our research lab.

B. Coding Dojo as a Learning Activity

The use of Coding Dojo has been adopted as an alternative to teaching programming languages and Test Driven Development. For instance, Heinonen et al. point out that coding dojo should be applied as an extracurricular activity, since it may not be suitable for some students [9]. On the other hand, Carnieto et al. discuss that Coding Dojo allow students to feel comfortable in making mistakes and learning from their errors, without the pressure of being evaluated [10]. This increases the degree of flexibility in relation to the learning goals.

Rodrigues et al. analyzed in two perspectives the results of implementing Coding Dojos as a learning strategy [3]. The Master's perspective combined with participants surveys demonstrate that students feel more comfortable and engaged in problem solving when applying Coding Dojos. In addition, this learn strategy encourage students to use new program practices. In contrast, the authors also report issues related to the difficulty of part of the students to program in front of the classmates.

B. Estacio et al. used Pair programming and Coding Dojo practices to investigate enthusiasm, user familiarity, and learning outcomes [1]. The authors report that, although Coding Dojo has offered constructive effects to the learning procedure, there are challenges related to motivation and participant experience.

The work presented by Schoeffel et al. [11] performs an experiment in the context of teaching object-oriented programming language, considering two treatment groups: the first used a traditional methodology and the second developed the discipline's activities using the Coding Dojo technique. The results obtained with the comparison showed that the group of students who participated in the second obtained significantly better grades than the group that did not use Coding Dojo sessions.

In another context, the work of Da Luz et al. [8] shows that the Coding Dojo technique can also be applied in the development of software tests. The authors conducted a questionnaire involving experienced professionals to collect qualitative information. In addition, they conducted an electronic research with Coding Dojo groups in Brazil, presenting the use of the

Fig. 1. Recent Coding Dojo sessions conducted in our research lab.

technique to support students to learn how to develop software with a high degree of test coverage.

Differently from existing research works, here we present the results of a long term experience (almost 18 months) using Coding Dojo, which let undergrad students to collaborate on a real effort of rejuvenating two non-trivial enterprise systems, including activities related to software requirements, design, implementation, test, and configuration management. In the next section we present more details about this software modernization effort.

C. Theoretical Pedagogical Framework

Our approach to use Coding Dojo as a pedagogical vehicle has as inspiration the *Problem-Based Learning* (PBL) method, which "involves confronting students with problems from practice, providing a stimulus for learning" [12]. PBL has emerged several years ago in the studies of Medicine, and is becoming a common approach to teach software engineering [13], [14], [15]. However, differently from existing works that report experiences on using PBL to teach software engineering in classroom [13], [14].

In this paper we present the learning experiences in Coding Dojo sessions that we carried out as complementary activities to the students. The collaboration environment that occurs during Coding Dojo sessions also enables a constructive approach based on *Communities of Practice* [16], which allow the students to develop not only technical skills, but also foster social and communication abilities.

III. Research Context

This work presents an experience report using the Coding Dojo technique in the course of a modernization process of two enterprise legacy systems. This research is part of a cooperation project between the University of Brasília (UnB) and the Center of System Development of the Brazilian Army. The first system (SISDOT) is a supplying management software for the materials and equipments used in the Brazilian Army, including vestments, automatic rifles, and combat vehicles.

These materials are most often of individual or collective use, and in some cases essential for the exercise of the activities of a Military Organization (MO). SISDOT is organized according to a typical JEE (Java Enterprise Edition) [17] architecture, using *Primefaces* for the presentation layer, the Java Beans and Context and Dependency Injection specifications for the service layer, and the Java Persistence API for the database layer. It includes five Maven modules, one for each business domain of the application and we use a single Git repository hosted on the Bitbucket cloud infrastructure.

There are two main reasons that motivate the modernization of SISDOT: migrate from a legacy architecture to the JEE platform and implement new features. The estimated size of the legacy SISDOT is around 750 function points, according to an independent consultant service.

The second system (SISBOL) deals with the official internal communication of the Brazilian Army. The core component of SISBOL is a flexible workflow that manages the process of creating parts of official documents (notes), composing notes into formal documents, and approving notes and formal documents for internal disclosure.

SISBOL is organized using three major architectural modules: a front-end module implemented in AngularJS; a back-end module implemented using a service-oriented architecture (on top of Java Enterprise Edition); and a set of micro-services (implemented using Spring Boot) that integrates SISBOL with a legacy *human-resources system*. The estimated size of the legacy SISBOL is around 700 function points, also according to an independent consultant service.

SISBOL and SISDOT share the same reasons that motivate the modernization effort. In this way, this research context deals with many facets related to software construction, such as requirements elicitation, reverse engineering, design, implementation and testing. Table I presents some figures about the

current implementation of these systems.

TABLE I
SOME FIGURES ABOUT THE CASE STUDIES (SISDOT AND SISBOL)

Project	Packages	Classes	Methods	LOC	LOC Test
SISDOT	62	453	2740	31998	8512
SISBOL	22	147	416	27215	11805

Table II shows the profile of the contributors of the modernization effort of the two systems, including degree and experience in other software development projects. The two systems are being developed in parallel.

IV. Research Goals and Questions

The general goal of our research is to understand how the participants perceive the use of Coding Dojo sessions (a) to facilitate the learning process of software development techniques and (b) to enable the sharing of experience between novices and practitioners. To this end we conducted a survey using an online questionnaire. Space constraints have obligated us to focus on a subset of the questions of our survey, which we consider most relevant to our general goal.

The questions we explore here are in what follows.

(Q1) Do you agree that the use of Coding Dojo favors the understanding of the software requirements?
(Q2) Do you agree that the use of Coding Dojo is effective to enable the development team to share knowledge?
(Q3) Do you agree that you have contributed to solve problems (either implementing or giving advices) during Coding Dojo sessions?
(Q4) Do you agree that you will feel more comfortable to start your career as a software engineering after your experience with Coding Dojo sessions?
(Q5) What software development disciplines (project management, requirements elicitation, architecture, design and implementation, tests) did you learn more during the Coding Dojo sessions?
(Q6) How Coding Dojo sessions have effected the modernization of the system you contribute to and how Coding Dojo sessions have contributed to bring you new skills?

Questions (Q1) — (Q4) are closed-ended questions using a Likert scale [18], ranging from *Strongly Disagree* (1) to *Strongly Agree* (5). Question (Q4) is a ranking question, where a participant should assign a value ranging from 1 to 5 for each available option—the value 5 corresponds to the software engineering subject she learned more through the Coding Dojo sessions and the value 1 corresponds to the software engineering discipline she learned less through the Coding Dojos. Question (Q5) is an open-ended question, where the participants could provide more detailed answers.

We conducted two forms of data analysis. In the first, we performed an exploratory data-analysis that leads to a general understanding of the closed-ended questions. We present the results of this data-analysis using either tables or plots. To the open-ended question, we highlight the main themes and quote here some of the relevant answers that might help to generalize the results of our survey.

V. Results and Discussion

We collected answers from 21 respondents and found results for questions (Q1)–(Q4) that encourage the adoption of Coding Dojo. Regarding the first question, 80.95% of the students agree that Coding Dojo sessions are relevant to discuss and understand the requirements of the modernization effort. For many reasons (including restricted access to the Brazilian Army Dependencies), the requirements elicitation activity for SISDOT is being conducted by a small and specialized team (one domain expert, one designer, and two software engineering practitioners) working using a *User Experience* approach.

Regarding SISBOL, the main source of requirements elicitation was based on reverse engineering of the legacy system, complemented by a couple of meetings with domain experts and designers. Therefore, the use of Coding Dojo sessions was a necessary step to understanding the requirements prior to the development activities. Based on the results of acceptance tests (conducted in the Brazilian Army divisions), we considered this approach effective, and the students working as software developers during Coding Dojo sessions were able to understand the requirements and implement the software components properly.

Regarding the second question, all participants agree that Coding Dojos contribute to knowledge sharing (28.57% agree and 71.42% strongly agree with the statement in (Q2)). This is particularly relevant in our context, because the Coding Dojo sessions often involved students collaborating with practitioners to solve real problems. In this way, we address one of the guidelines of the ACM / IEEE Software Engineering Curriculum, which recommends "*student experiences with the professional practice of software engineering*" [19].

Considering the third question, 90.46% of the respondents either agree (19.04%) or strongly agree (71.42%) that they have contributed to the Coding Dojo sessions positively, against 9.54% that neither agree nor disagree about the relevance of their contribution. As an *extra question*, we asked the students whether or not the format of our Coding Dojo sessions is adequate. We find answers that reveal some *preparation* and *execution* flaws. For instance, some respondents make clear that, in specific sessions, part of the participants did not have the opportunity to contribute to the source code. That is, the *15 to 20 minutes* rule for each turn was not always respected. Other participants complain about the lack of a public agenda for the Coding Dojo sessions. In addition, some planned tasks were too complex to be solved during a Coding Dojo, and the absence of a software architect in specific sessions could have compromised part of the results.

Although we have tried to avoid competition during the Coding Dojo sessions, some students also report that certain degree of competition arise during our Coding Dojo sessions, which might have inhibited the contribution of new participants. All these aspects deserve more attention in future activities, though, based on the answers to questions (Q2) and

TABLE II
CHARACTERIZATION OF THE PROJECT'S PARTICIPANTS

Role	Id	Project	Degree	Experience
Project Manager	PM	SISDOT and SISBOL	PhD	18 years
Software Architect	SA1	SISBOL	graduated	10 years
Software Architect	SA2	SISDOT	graduated	14 years
Software Architect	SA3	SISDOT	Master Student	15 years
Designer	DES1	SISDOT and SISBOL	Graduated	7 years
Designer	DES2	SISDOT and SISBOL	Undergraduate Student	2 years
Developer	DEV1	SISDOT	Master Student	02 years
Developer	DEV2	SISDOT	Master Student	02 years
Developer	DEV3	SISDOT	Master Student	05 years
Developer	DEV4	SISBOL	Master Student	01 years
Intern	I1	SISBOL	Undergraduate Student	1 year
Intern	I2	SISBOL	Undergraduate Student	6 months
Intern	I3	SISBOL	Undergraduate Student	until 01 year
Intern	I4	SISBOL	Undergraduate Student	until 01 year
Intern	I5	SISBOL	Undergraduate Student	until 01 year
Intern	I6	SISBOL	Undergraduate Student	until 01 year
Intern	I7	SISBOL	Undergraduate Student	until 01 year
Intern	I8	SISDOT	Undergraduate Student	until 01 year
Intern	I9	SISDOT	Undergraduate Student	until 01 year
Intern	I10	SISDOT	Undergraduate Student	until 01 year
Intern	I11	SISDOT	Undergraduate Student	until 01 year

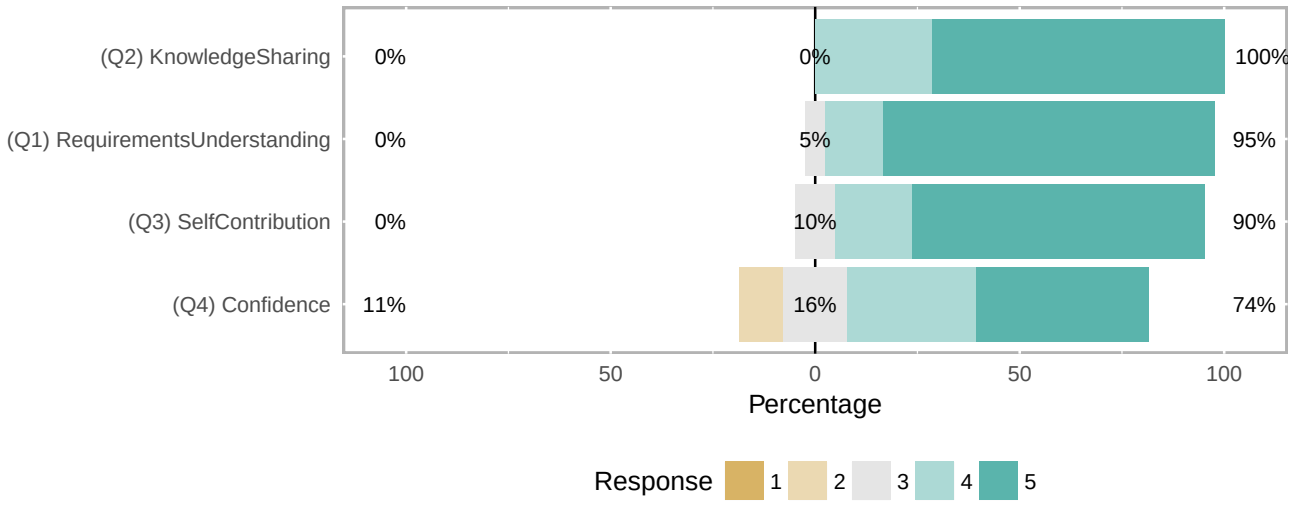

Fig. 2. Answers' distribution for questions (Q4)–(Q5)

(Q3), and the current status of the modernization effort, these flaws have not compromised the learning and development outcomes.

With respect to the fourth research question, which we aim to understand whether or not Coding Dojos increase the confidence of the students to assume professional positions, we found that 10% of the students disagree about the statement, and 15% neither agree nor disagree with that—against 73% that either agree or strongly agree. That is an interesting result, particularly because, considering two additional questions, 80% of the students agree that the use of Coding Dojos improve general skills related to software architecture and software development tools and 70% agree that Coding Dojos improve programming language skills. These results complement each other and show that, for 20% to 30% of the students, Coding Dojo is not effective to teach technical aspects related to software architecture, software development tools, and programming languages.

Also regarding (RQ4), the results (almost 26% do not increase their confidence to start working as a practitioner through the Coding Dojo sessions) might have been motivated because some of the participants are at the first years of their undergraduate course, and thus might still feel uncomfortable in taking professional activities. The results so far, that we summarize in Figure 2 and Table III, lead to the following interpretation.

Summary of (Q1)-(Q4): We found evidences that Coding Dojo is an effective approach for sharing experiences among novices and practitioners, which contributes to the students education. However, to be effective, Coding Dojo sessions must be planned in advance and the rules must be strictly obeyed. This might be a challenge, particularly when conducted in real settings. Surprisingly, for 30% of the participants, Coding Dojo sessions do not improve some technical skills that have being explored in our research context.

To answer Question (Q5), which aims to investigate what software engineering fields the participation in Coding Dojo promote more knowledge acquisition, we use a variant of the Paired Comparison Method [20] available in the `prefmod R package` [21]. Based on a ranking question, this method estimates the preference of each option—here, a software engineering field, including *Project and Configuration Management*, *Requirements Elicitation*, *Architectural Design*, *Software Constructions*, and *Automated Tests*. This is an optional question of our survey, and thus we collected information from 13 out of 21 participants of the survey.

Table IV presents the answers to this research question. According to the respondents, Software Construction and Architectural Design are the software engineering fields the participants learned more through our Coding Dojo sessions. Software construction was the essence of many Coding Dojo sessions and the students might have also favored software architecture because, during a few weeks, we conducted several dojos focusing on the modularization of a group of features using a microservices-based architecture [22], [23].

In that period, we believe that the students learned several concepts related to this architectural style. Automated Testing and Requirements Elicitation gained the same preference. We considered this result quite surprising, since we explored several libraries and tools for Automated Testing (including JUnit, Cucumber, and Selenium), and spent a significant amount of time working with Test Driven Development [24]. Perhaps, the frontier between Software Construction and Automated Testing was not so clear to the students.

Project and Configuration Management was the software engineering field which the students considered having learning a slighter amount through Coding Dojos. We followed an agile approach, planing each sprint in a collaborative way. Each sprint should design, implement, test, and integrate a set of features and user stories [25], [26]. During a sprint, our recommendation was to follow a feature-branch approach for configuration management, where developers should create a specific branch for each new functionality or bug-fix. Most of the students learned how to proceed in these activities, including tasks for creating new branches, integrating the results to the main development branch, submitting pull-requests for further analysis, and so on. We believe that this was one of the most successful learning aspects of Coding Dojos, though, for some reason, the students do not perceived as such.

Summary of (Q5): We found evidences that the students perceived themselves as learning more Software Construction and Architectural Design through our Coding Dojos, when compared to other software engineering disciplines (Project and Configuration Management, Requirements Elicitation, and Automated Testing). Our perceptions is that the students learned a lot of Configuration Management and Automated Testing as well (even more than Architectural Design), though, for some reason, they do not perceived as such.

Finally, to answer the sixth question *How Coding Dojo sessions have effected the modernization of the system you contribute to and how Coding Dojo sessions have contributed to bring you new skills?*, we first identified common themes in the answers and then quote here representative ones. Regarding the first part of the question (*benefits to the systems' development*), we identified six frequent themes in the responses (see Table V), though Knowledge Sharing appears in 37.5% of the responses. For instance, one of the respondents answered that Coding Dojos "*...unify the team and help to spread knowledge along everyone in the team"*. Surprisingly, four answers (16.66%) mentioned that Coding Dojo increases the productivity of the teams. For instance, one of the respondents state that "*The productivity of some members have significantly improved after the adoption of Coding Dojos. Besides that, Coding Dojo was essential to introduce new developers to the project*".[1]

We were not expecting this benefit when we started to conduct Coding Dojos. Two answers also mentioned that Coding Dojo sessions help to identify good solutions for solving the problems, as one of the participants makes clear that Coding Dojos "*helped a lot to discuss the problems with the participants and find solutions together. Often these solutions are better than those found alone.*"

Regarding the second part of the question (*how Coding Dojo sessions have contributed to bring you new skills?*), we identified 4 recurrent themes in the answers (see Table VI), highlighting which personal aspects of a software engineer have been increased using Coding Dojos. The two most frequent themes relate to gains in Technical Skills (appearing in 38.46% of the answers) and Collaborative Work (appearing in 30.76% of the answers).

For instance, one of the respondents stated that Coding Dojos "*...reduced the learning curve of some tools, and contributed to familiarize myself to the existing source code and the JEE architecture*", another explained that Coding Dojo enabled her to "*...learn that I didn't have to fear or get shy about contributing (to the source code), even without having so much experience. I used to be really insecure when it was my turn to code. Now, I understand that we are there to learn and gain experience, so we need to take advantage of the best of each other to work as a team*", and finally another student

[1]We marked two themes in this answer (*Increasing Productivity* and *Team Integration*)

TABLE III
DESCRIPTIVE STATISTICS FOR QUESTIONS (Q1)–(Q4)

Item	Mean	Standard Deviation	Low	Neutral	High
(Q1) Requirements Understanding	4.76	0.54	0.00	4.76	95.24
(Q2) Knowledge Sharing	4.71	0.46	0.00	0.00	100.00
(Q3) Self Contribution	4.62	0.67	0.00	9.52	90.48
(Q4) Confidence	4.05	1.03	10.53	15.79	73.68

TABLE IV
ANSWERS TO THE RESEARCH QUESTION (RQ5)

Subject Id	Project and Configuration Management	Requirements Elicitation	Architectural Design	Software Construction	Automated Testing
1	3	5	4	4	3
2	2	5	4	5	1
3	4	5	5	5	5
5	4	4	2	2	2
6	3	4	4	4	5
7	3	4	4	5	1
9	3	3	5	4	5
10	5	2	4	3	2
11	4	5	4	4	4
12	4	5	5	5	4
13	2	2	3	3	4
14	2	3	5	5	2
21	3	3	3	4	2
Estimated Preference	0.13	0.15	0.26	0.31	0.15

TABLE V
FREQUENT TERMS RELATED TO THE QUESTION HOW CODING DOJO SESSIONS HAVE EFFECTED THE MODERNIZATION OF THE SYSTEM YOU CONTRIBUTE TO?

Term	Count	Frequency (%)
Sharing Knowledge	9	37.5%
Increasing Productivity	4	16.66%
Requirements Understanding	3	12.5%
Interesting Technical Decisions	2	8.33%
Team Integration	2	8.33%
Problem Solving	2	8.33%

claimed that "*dojos helped me as a developer to understand the importance of automatic tests, and learning how to teach properly some subjects to everyone*". Surprisingly, only 7.69% of the answers relate to *Learning Programming Language Skills*, though this benefit has been discussed as one of the main expected outcomes of using Coding Dojo sessions in academic settings [1], [3]. As one example, one respondent states that "*I didn't know Java as well as my coworkers, and I believe everyone learned just as I did*".

TABLE VI
FREQUENT TERMS RELATED TO THE QUESTION: HOW CODING DOJO SESSIONS HAVE CONTRIBUTED TO BRING YOU NEW SKILLS?

Term	Count	Frequency (%)
Improved Technical Skills	10	38.46%
Improved Collaborative Work Skills	8	30.76%
Improved Problem Solving Skills	3	11.53%
Learning Programming Language Skills	2	7.69%

Summary of (Q6): We found evidences that the students understand that Coding Dojos foster Knowledge Sharing and Increase the Productivity of the developers. Surprisingly, only 7.69% of the answers relate to the personal benefit of increasing programming language skills.

VI. THREATS TO VALIDITY

In this paper we detailed the results of a qualitative study to *understand the understanding of students* about the benefits that Coding Dojos bring to the learn process of software engineering and to the modernization of two enterprise legacy systems. This type of research presents some threats to generalize (external validity) the results to other contexts. For instance, here we detail a 18 months experience working with the same set of methods, techniques, and programming languages.

We cannot generalize our results to other settings. Nevertheless, it is important to note that we have been working with students of three undergraduate courses (Computer Science, Mechatronics, and Software Engineering) using a *state-of-the-practice* (a) development model (including test-driven and feature branch development) and (b) architecture (including AngularJS, Java Enterprise Edition on top of CDI, and microservices). In this way, we believe that our findings are relevant to both academy and industry.

Regarding construct validity, the analysis of the sixth question of our survey led to a quite surprising result. The answers to this question diverge from our personal observations: we used to we believe that the students learned the subjects Configuration Management and Automated Testing as much as

they learned the subject Software Constructions. It is still not clear to us the reason for this divergence. One possible option is that our personal observations are wrong. Another option is that this particular question was not well designed, leading to a wrong conclusion. Since the divergence is significant, we believe that our personal observations are wrong. As a future work, we want to further investigate this question.

VII. Conclusion

In this paper we presented an experience report of using Coding Dojos to favor the learning process of software engineering practices—conducted in the context of a real, non-trivial modernization effort of two enterprise systems of the Brazilian Army.

Differently from other studies that investigate the use of coding dojos in academic settings, our investigation was conducted "in the wild", with strong expectations and using state-of-the-practices and technologies, allowing the students to have an experience close to what is expected in the software engineering industry.

Based on a qualitative study, we observed that the use of Coding Dojo brings several benefits to the students (e.g., it improves technical and collaborative skills) and to the project development (fostering knowledge sharing among the participants and productivity).

Acknowledgment

The authors would like to thank the Brazilian Army for their support to this work and to allow our University to conduct this research in the context of a real software modernization effort (FUB / CIC 6138/2016).

References

[1] B. Estácio, R. Oliveira, S. Marczak, M. Kalinowski, A. Garcia, R. Prikladnicki, and C. Lucena, "Evaluating collaborative practices in acquiring programming skills: Findings of a controlled experiment," in *2015 29th Brazilian Symposium on Software Engineering*, Sept 2015, pp. 150–159.

[2] E. Bache, "The coding dojo handbook," *Emily Bache, Canada*, 2013.

[3] P. L. da R. Rodrigues, L. P. Franz, J. F. P. Cheiran, J. a. P. S. da Silva, and A. S. Bordin, "Coding dojo as a transforming practice in collaborative learning of programming: An experience report," in *Proceedings of the 31st Brazilian Symposium on Software Engineering*, ser. SBES'17. New York, NY, USA: ACM, 2017, pp. 348–357. [Online]. Available: http://doi.acm.org/10.1145/3131151.3131180

[4] T. Erl, *Soa: principles of service design*. Prentice Hall Upper Saddle River, 2008, vol. 1.

[5] N. Jain, A. Bhansali, and D. Mehta, "Angularjs: A modern mvc framework in javascript," *International Journal of Global Research in Computer Science (UGC Approved Journal)*, vol. 5, no. 12, pp. 17–23, 2015.

[6] D. T. Sato, H. Corbucci, and M. V. Bravo, "Coding dojo: An environment for learning and sharing agile practices," in *Agile, 2008. AGILE'08. Conference*. IEEE, 2008, pp. 459–464.

[7] B. J. d. S. Estácio *et al.*, "Uma avaliação empírica sobre a aprendizagem colaborativa em coding dojo randori no contexto de desenvolvimento de software," 2017.

[8] R. B. Da Luz, A. G. S. S. Neto, and R. V. Noronha, "Teaching tdd, the coding dojo style," in *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*. IEEE, 2013, pp. 371–375.

[9] K. Heinonen, K. Hirvikoski, M. Luukkainen, and A. Vihavainen, "Learning agile software engineering practices using coding dojo," in *Proceedings of the 14th annual ACM SIGITE conference on Information technology education*. ACM, 2013, pp. 97–102.

[10] A. P. Carnieto, G. F. da Silva, I. M. de Lima Bicudo, M. S. de Souza, A. G. J. Lan, A. M. M. Funabashi, P. H. F. Cassiano, and S. M. Peres, "Competec-dojo de programação como reforço ao ensino de programação para alunos do ensino médio técnico," *Revista ComInG-Communications and Innovations Gazette*, vol. 2, no. 1, pp. 1–12, 2017.

[11] P. Schoeffel, D. F. Rosa, and R. S. Waslawick, "Um experimento do uso de coding dojo na aprendizagem de programação orientada a objetos," *iSys-Revista Brasileira de Sistemas de Informação*, vol. 9, no. 2, 2016.

[12] D. Boud and G. Feletti, *The challenge of problem-based learning*. Psychology Press, 1997.

[13] J. D. Delaney, G. Mitchell, and S. Delaney, "Software engineering meets problem-based learning," *The Engineers Journal*, vol. 57, no. 6, 2003.

[14] M. J. Oudshoorn and K. J. Maciunas, "Experience with a project-based approach to teaching software engineering," in *Software Education Conference, 1994. Proceedings.*, Nov 1994, pp. 220–225.

[15] A. Fox, D. A. Patterson, and S. Joseph, *Engineering software as a service: an agile approach using cloud computing*. Strawberry Canyon LLC, 2014.

[16] E. Wenger, *Communities of Practice: Learning, Meaning, and Identity*. Cambridge University Press, 1998.

[17] E. Jendrock, R. Cervera-navarro, I. Evans, K. Haase, and W. Markito, "Java platform, enterprise edition (the java ee tutorial)," 2014.

[18] I. E. Allen and C. A. Seaman, "Likert scales and data analyses," *Quality progress*, vol. 40, no. 7, p. 64, 2007.

[19] R. LeBlanc and A. Sobel, "Curriculum guidelines for undergraduate degree programs in software engineering," Joint Task Force on Computing Curricula IEEE Computer Society Association for Computing Machinery, Tech. Rep., 2015.

[20] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: I. the method of paired comparisons," *Biometrika*, pp. 324–345, 1952.

[21] R. Hatzinger and M. J. Maier, *prefmod: Utilities to fit paired comparison models for preferences*, 2014, R package version 0.8-32. [Online]. Available: http://CRAN.R-project.org/package=prefmod

[22] J. Lewis and F. Martin, "Microsservices," http://martinfowler.com/articles/microservices.html, 2014, accessed: 2017-07-01.

[23] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on*. IEEE, 2016, pp. 44–51.

[24] K. Beck, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[25] K. Schwaber and M. Beedle, *Agile software development with Scrum*. Prentice Hall Upper Saddle River, 2002, vol. 1.

[26] M. Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.